Tartalom

DP25a - 5. FP gyakorlat, 2025-10-21 **1**
Bináris fák feldolgozása elágazó rekurzióval, 2. rész 1
T8. Bal és jobb szélső címkék visszaadása 1
T9. Bináris fa rendezettsége 2
T10. Bináris fa összes címkéjének útvonala 2
T11. Adott címke összes előfordulása bináris fában útvonallal 3

DP25a - 5. FP gyakorlat, 2025-10-21

Bináris fák feldolgozása elágazó rekurzióval, 2. rész

A feladatokhoz az alábbi tree adattípust definiáljuk:

```
@type tree() :: :leaf | {any(), tree(), tree()}
```

Tehát egy tree() típusú Elixir-kifejezés

- vagy egy adatot (a továbbiakban *címkének* nevezzük) tartalmazó *csomópont*, amely további két tree() típusú értéket tartalmaz: az első a bal, a második a jobb részfa;
- vagy egy címke nélküli :leaf (azaz *levél)* atom.

A példákban felhasznált változók és értékük:

```
t1 = {4,
      {3,:leaf,:leaf},
      {6,
        {5,:leaf,:leaf},
        {7,:leaf,:leaf}
      }
    }
t2 = {:a,
       {:b, {:v,:leaf,:leaf}, :leaf},
       {:c,
          :leaf,
          {:d,
             {:w, {:x,:leaf,:leaf}, :leaf},
             {:f, {:x,:leaf,:leaf}, {:y,:leaf,:leaf}}
          }
       }
    }
```

A típust és a két változó helyett a két függvényt definiáljuk a T modulban, hogy más modulokból hivatkozhassunk rájuk:

```
defmodule T do
  @type tree() :: :leaf | {any(), tree(), tree()}
  @type inttree() :: :leaf | {integer(), inttree(), inttree()}
  def t1() do
    {4, {3, :leaf, :leaf}, {6, {5, :leaf, :leaf}, {7, :leaf, :leaf}}}
  end

  def t2() do
    {:a, {:b, {:v, :leaf, :leaf}, :leaf},
     {:c, :leaf,
      {:d, {:w, {:x, :leaf, :leaf}, :leaf}, {:f, {:x, :leaf, :leaf}, {:y, :leaf, :leaf}}}}}
  end
end

IO.puts("t1 = " <> inspect(T.t1()))
IO.puts("t2 = " <> inspect(T.t2()))
```

T8. Bal és jobb szélső címkék visszaadása írjon egy-egy függvényt egy bináris fa bal, ill. jobb szélső címkéjének visszaadására!

```
defmodule MostLeftRight do
 @spec fa_balerteke(f::T.tree()) :: {:ok, c::any()} | :error
 # Egy nemüres f fa bal oldali szélső címkéje c (minden
 # felmenőjére is igaz, hogy bal oldali gyermek)
 # Ha nincs bal oldali szélső érték, az :error atomot adja vissza
 def fa_balerteke(f) do
 ...
 end

 @spec fa_jobberteke(f::T.tree()) :: {:ok, c::any()} | :error
 # Egy nemüres f fa jobb oldali szélső címkéje c (minden
 # felmenőjére is igaz, hogy jobb oldali gyermek)
 # Ha nincs jobb oldali szélső érték, az :error atomot adja vissza
 def fa_jobberteke(f) do
 ...
 end
end
(MostLeftRight.fa_balerteke(T.t1()) === {:ok, 3}) |> IO.inspect()
(MostLeftRight.fa_balerteke(:leaf) === :error) |> IO.inspect()
(MostLeftRight.fa_jobberteke(T.t1()) === {:ok, 7}) |> IO.inspect()
(MostLeftRight.fa_jobberteke(:leaf) === :error) |> IO.inspect()
```

T9. Bináris fa rendezettsége Egy bináris fa rendezett, ha *inorder* bejárásakor a címkéi *szigorúan monoton növekednek*, azaz a csomópontjai kielégítik a keresőfa-tulajdonságot: minden egyes csomópont címkéje nagyobb a bal oldali gyermekei címkéjénél és kisebb a jobb oldali gyermekei címkéjénél. Oldja meg a feladatot a MostLeftRight.fa_balerteke/1 és a MostLeftRight.fa_jobberteke/1 függvényekkel.

Megírhatja a 4. gyakorlaton szerepelt Labels.cimkek/1, valamint az Enum.sort/1 vagy más függényekkel (pl. saját segédfüggénnyel) is.

```
defmodule Ordered do
 @spec rendezett(f::T.tree()) :: b::boolean()
 # b igaz, ha az f fa rendezett
 def rendezett(f) do
 ...
 end
end
(Ordered.rendezett(T.t1()) === true) |> IO.inspect()
(Ordered.rendezett(T.t2()) === false) |> IO.inspect()
```

T10. Bináris fa összes címkéjének útvonala Egy adott csomópont útvonalának nevezzük azon csomópontok címkéinek listáját, amelyeken át a fa gyökerétől az adott csomópontig el lehet jutni.

Javasoljuk a Route.utak/2 segédfüggvény definiálását és használatát.

```
defmodule Route do
 @type route() :: [any()]
 @spec utak(f::T.tree()) :: cimkezett_utak::[{c::any(), cu::route()}]
 # A cimkezett_utak lista az f fa minden csomópontjához egy {c,cu} párt
 # társít, ahol c az adott csomópont címkéje, cu pedig az adott
 # csomóponthoz vezető útvonal
 def utak(f) do
 ...
 end

 @spec utak(f::T.tree(), eddigi_ut::route()) :: cimkezett_utak::[{c::any(), cu::route()}]
 # A cimkezett_utak lista az f fa minden csomópontjához egy {c,cu} párt
 # társít, ahol c az adott csomópont címkéje, cu pedig az adott
 # csomóponthoz vezető útvonal az eddigi_ut eddigi útvonal elé fűzve
 def utak(f) do
 ...
 end
end
(Route.utak(T.t1()) === [{4,[]},{3,[4]},{6,[4]},{5,[4,6]},{7,[4,6]}]) |> IO.inspect()
(Route.utak(T.t2()) === [{:a,[]},
```

```
{:b,[:a]},
{:v,[:a,:b]},
{:c,[:a]},
{:d,[:a,:c]},
{:w,[:a,:c,:d]},
{:x,[:a,:c,:d,:w]},
{:f,[:a,:c,:d]},
{:x,[:a,:c,:d,:f]},
{:y,[:a,:c,:d,:f]}
]) |> IO.inspect()
```

T11. Adott címke összes előfordulása bináris fában útvonallal Oldja meg a feladatot

1. for-jelöléssel és a Route.utak/1 függvény felhasználásával;
2. takarékoskodva a memóriával, azaz az összes útvonal helyett csak a keresett útvonalakat tárolja el. (A megoldásban a Route.utak/2 függvényhez hasonló segédfüggvényt használjon, de a gyökér címkéjét csak egy bizonyos feltétel teljesülésekor tárolja el.)

```
defmodule Occurrences do
 @spec cutak(f::T.tree(), c::any()) :: utak::[{c::any(), cu::Route.route()}]
 # utak azon csomópontok útvonalainak listája f-ben, amelyek címkéje c
 def cutak(f) do
 ...
 end
end
(Occurrences.cutak(T.t1(), :x) === []) |> IO.inspect()
(Occurrences.cutak(T.t2(), :x) === [{:x,[:a,:c,:d,:w]},{:x,[:a,:c,:d,:f]}]) |> IO.inspect()
```